INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Stamp / Signature of the Invigilator

EXAMINATION (End Semester)										**SEMESTER (Autumn)**			
Roll Number										**Section**		**Name**	
Subject Number	C	S	6	0	0	2	6	**Subject Name**		*Parallel and Distributed Algorithms*			
Department / Center of the Student											**Additional sheets**		

Important Instructions and Guidelines for Students

1. You must occupy your seat as per the Examination Schedule/Sitting Plan.
2. Do not keep mobile phones or any similar electronic gadgets with you even in the switched off mode.
3. Loose papers, class notes, books or any such materials must not be in your possession, even if they are irrelevant to the subject you are taking examination.
4. Data book, codes, graph papers, relevant standard tables/charts or any other materials are allowed only when instructed by the paper-setter.
5. Use of instrument box, pencil box and non-programmable calculator is allowed during the examination. However, exchange of these items or any other papers (including question papers) is not permitted.
6. Write on both sides of the answer script and do not tear off any page. **Use last page(s) of the answer script for rough work.** Report to the invigilator if the answer script has torn or distorted page(s).
7. It is your responsibility to ensure that you have signed the Attendance Sheet. Keep your Admit Card/Identity Card on the desk for checking by the invigilator.
8. You may leave the examination hall for wash room or for drinking water for a very short period. Record your absence from the Examination Hall in the register provided. Smoking and the consumption of any kind of beverages are strictly prohibited inside the Examination Hall.
9. Do not leave the Examination Hall without submitting your answer script to the invigilator. **In any case, you are not allowed to take away the answer script with you.** After the completion of the examination, do not leave the seat until the invigilators collect all the answer scripts.
10. During the examination, either inside or outside the Examination Hall, gathering information from any kind of sources or exchanging information with others or any such attempt will be treated as **'unfair means'**. Do not adopt unfair means and do not indulge in unseemly behavior.

Violation of any of the above instructions may lead to severe punishment.

Signature of the Student

To be filled in by the examiner

Question Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Total**
Marks Obtained											

Marks obtained (in words)	**Signature of the Examiner**	**Signature of the Scrutineer**

CS60026 Parallel and Distributed Algorithms, Autumn 2017–2018

End-Semester Test

22–November–2017 CSE 107/108/119, 09:00–12:00 hours Maximum marks: 60

Roll no: _____ **Name:** _____

[*Write your answers in the question paper itself. Be brief and precise. Answer all questions.*
If you use any algorithm/result/formula covered in the class, just mention it, do not elaborate.]

1. Suppose that a parallel algorithm for CREW PRAMs runs in $T(n)$ time and does $W(n)$ work. Prove that a ***p*-processor EREW PRAM** can run the same algorithm in $O\left(\left(\frac{W(n)}{p}+T(n)\right)\log p\right)$ time. **(10)**

Solution By the WT scheduling principle, the algorithm runs in $\frac{W(n)}{p}+T(n)$ time on a p-processor CREW PRAM. In order to run the algorithm on a p-processor EREW PRAM, we need to handle the concurrent read instructions. The number r of processors that attempt to read concurrently from a location x at a point of time is $\leqslant p$. Let us see how an array $X=(x_1,x_2,\ldots,x_r)$ with all $x_i=x$ can be prepared in $O(\log r)$ (that is, $O(\log p)$) time without using concurrent reads. The following pseudocode assumes for simplicity that $r=2^t$.

```
Processor 1 reads x and copies the value to x_1.
For i = 0,1,2,...,t-1, repeat:
    For j = 1,2,...,2^i, pardo:
        Processor j copies x_j to x_{2^i+j}.
```

After this, an r-way concurrent reading of x is replaced by exclusive reading of $x_1,x_2,\ldots,x_r$ by r processors from the different locations in X. Since the maximum number of concurrent reads is $\frac{W(n)}{p}+T(n)$ and each concurrent read expands to $O(\log p)$ time, the result follows.

2. Let $A = (a_1, a_2, \ldots, a_n)$ be a sorted array of n integers. You are given an **EREW PRAM** with ***p* processors** and an (arbitrary) array $B = (b_1, b_2, \ldots, b_p)$ of integers. The task of the i-th processor is to search for b_i in A, and to write the result c_i (true/false) in an array $C = (c_1, c_2, \ldots, c_p)$. Assume that $p = \mathrm{O}(\log n)$, and that n is a power of 2. Exercise 1 leads to an $\mathrm{O}(\log n \log\log n)$-time solution to this problem. Propose an $\mathrm{O}(\log n)$-time algorithm for solving this problem. **(10)**

Solution The idea is to use pipelining. Take $n = 2^t$. Let us look into the array locations accessed during different iterations of the binary-search algorithm. In the first iteration, the middle element at index 2^{t-1} is accessed. In the second iteration, the element at one of the indices 2^{t-2} and $3 \times 2^{t-2}$ is accessed. In the third iteration, the element at one of the indices 2^{t-3}, $3 \times 2^{t-3}$, $5 \times 2^{t-3}$ and $7 \times 2^{t-3}$ is accessed, and so on. Evidently, there cannot be any conflict between the reads of two different iterations. Therefore if the i-th processor starts the binary search for b_i in A at time i, there will not be any concurrent read requests, and the processors finish all of the p searches in $\leqslant p + \log n$ time. Since $p = \mathrm{O}(\log n)$, this time is $\mathrm{O}(\log n)$.

3. Propose an $O(\log n)$-time algorithm to sort an array of n integers on a **priority CRCW PRAM**. Do not use the pipelined merge sort algorithm for CREW PRAMs, partially covered in the class. **(10)**

Solution Let X and Y be two sorted arrays of size m each. It suffices to show that $\text{Rank}(X;Y)$ can be computed by a priority CRCW PRAM in $O(1)$ time. We use m^2 processors indexed by the pair $(i,j) \in \{1,2,\ldots,m\}^2$. The processors are given priority based on the lexicographic ordering of the pairs (i,j) (or on the integer $mi+j$). Processor (i,j) is of higher priority than Processor (i',j') if (i,j) is lexicographically smaller than (i',j') (or equivalently $mi+j < mi'+j'$). The following code snippet computes the ranks of all x_i in Y in an array R.

```
For all i = 1,2,...,m, pardo:
    Initialize R[i] = m.
For all i = 1,2,...,m and for all j = 1,2,...,m, pardo:
    If x_i < y_j, set R[i] = j - 1.   /* To be done by Processor (i,j) */
```

The initialization step does not require any concurrent writes. So let us look at the second parallel step. Fix some $i \in \{1,2,\ldots,m\}$. Only the processors (i,j) with $j \in \{1,2,\ldots,m\}$ may attempt to write to $R[i]$. Let $\text{Rank}(x_i;Y) = r$, that is, $y_1 < y_2 < \cdots < y_r < x_i < y_{r+1} < y_{r+2} < \cdots < y_m$. For $j \leqslant r$, the condition $x_i < y_j$ does not hold, and Processor (i,j) does not make an attempt to write to $R[i]$. The condition $x_i < y_j$ holds for all $j = r+1, r+2, \ldots, m$, and so all the processors (i,j) with j in this range attempt to write to $R[i]$. Among these, Processor $(r+1,j)$ has the highest priority, and succeeds in writing $(r+1)-1 = r$ at $R[i]$.

Since merging two arrays can be done in $O(1)$ time, the running time of parallel merge sort on a priority CRCW PRAM satisfies $T(n) = T(n/2) + O(1) = O(\log n)$.

4. A **sum CRCW PRAM** resolves write conflicts by writing the sum of the values of all concurrent write requests. Solve the following problem in $O(1)$ time on a sum CRCW PRAM.

Let $A = (a_1, a_2, \ldots, a_n)$ be an unsorted array of $n = 2k$ integers. It is given that exactly half (that is, k) of the elements of A are positive, and the remaining k elements are negative. The problem is to prepare an array B consisting of all the elements of A such that the positive and negative elements of A are positioned alternately in B, and B starts with a positive element. Your algorithm should preserve the order of the positive and negative elements as they appear in A. For example, if $A = (-7, 2, -1, -9, 1, 4, -1, 5)$, then B should be $(2, -7, 1, -1, 4, -9, 5, -1)$. **(10)**

Solution Let $X = (x_1, x_2, \ldots, x_n)$ be an array of n integers. The following algorithm correctly computes the prefix sums of X in an array $XP = (xp_1, xp_2, \ldots, xp_n)$ in $O(1)$ time on a sum CRCW PRAM.

For $i = 1, 2, \ldots, n$ **and for** $j = i, i+1, \ldots, n$, **pardo: Write** x_i **to** xp_j.

Now, we solve the given problem. We use two arrays P and N to locate the positions of the positive and negative elements of A. This can be done in $O(1)$ time. We then compute the prefix sums of P and N in two arrays PP and NP in $O(1)$ time. Finally, guided by these prefix-sum arrays, we write appropriate elements of A to the desired positions in B, again in $O(1)$ time.

For $i = 1, 2, \ldots, n$, **pardo**
  If $a_i > 0$, **set** $P[i] = 1$ **and** $N[i] = 0$, **else set** $P[i] = 0$ **and** $N[i] = 1$.
Compute in parallel the prefix sums of P **in** PP **and of** N **in** NP.
For $i = 1, 2, \ldots, n$, **pardo**
  If $P[i] = 1$, **copy** a_i **to** $B[2 \times PP[i] - 1]$, **else copy** a_i **to** $B[2 \times NP[i]]$.

5. Let $V = \{1,2,3,\ldots,n\}$, and $D : V \to V$ a function. We know that D defines a pseudoforest on V (that is, a vertex-disjoint collection of pseudotrees, where a pseudotree is a rooted tree with an extra edge). Supplied additionally as input, an integer key value $K(i)$ for each Vertex i. Your task is to compute, for all i, the smallest key value stored in the pseudotree that contains Vertex i. Propose an $O(\log n)$-time algorithm for solving this problem on a **CREW PRAM** (write the complete pseudocode of your algorithm). What is the total work done by your algorithm? **(10)**

Solution We use the pointer jumping technique. We first copy the array D to an array P. The minimum for Vertex i is stored in $M(i)$.

```
For i = 1,2,...,n, pardo:
    Copy D(i) to P(i).
For i = 1,2,...,n, pardo:
    Initialize M(i) = K(i).
    Repeat ⌈log n⌉ times:
        If M(P(i)) < M(i), set M(i) = M(P(i)).
        Set P(i) = P(P(i)).
```

The total work done at this stage is $O(n \log n)$.

The above algorithm works on a pseudotree if and only if the minimum in the pseudotree lies on its cycle. There is more to be done in order to make the idea work. This calls for $O(n^2)$ space and an additional $O(n^2)$ work.

Each vertex calculates the minimum of its children, and replaces its minimum by the child minimum if needed.

```
For i = 1,2,...,n and for j = 1,2,...,n pardo:
    Set A(i,j) = ∞.
For j = 1,2,...,n pardo:
    If P(j) = i, set A(i,j) = M(j).
For each i = 1,2,...,n, pardo:
    Compute the minimum m_i of A(i,j), j = 1,2,...,n.   /* O(log n) time on CREW PRAM */
    If m_i < M(i), set M(i) = m_i.
```

Now, restore the original parent pointers, and do pointer jumping once more (another $O(n \log n)$ work).

```
For i = 1,2,...,n, pardo:
    Copy D(i) to P(i).
For i = 1,2,...,n, pardo:
    Repeat ⌈log n⌉ times:
        If M(P(i)) < M(i), set M(i) = M(P(i)).
        Set P(i) = P(P(i)).
```

6. The **Linear Equalities (LE)** problem is defined as follows. Let A be an $m \times n$ matrix with integer entries, and $\mathbf{b}$ an m-dimensional (column) vector. The problem is to decide whether there exists an n-dimensional vector $\mathbf{x} \geqslant \mathbf{0}$ such that $A\mathbf{x} = \mathbf{b}$. Prove that LE is P-Complete. **(10)**

Solution We can check the solvability of a system of linear equations in polynomial time by linear-algebra algorithms (like Gaussian elimination). Therefore LE is in P.

In order to prove the P-hardness of LE, we use reduction from LI (Linear Inequalities) which has been proved to be P-Complete. Let $c_1y_1 + c_2y_2 + \cdots + c_ky_k \leqslant d_l$ be a linear inequality (an inequality in the input instance of LI). We introduce a new variable z_l (called a slack variable) to convert this inequality to the equality $c_1y_1 + c_2y_2 + \cdots + c_ky_k + z_l = d_l$ along with the condition $z_l \geqslant 0$. Clearly, all the inequalities can be converted to equalities in $\mathrm{O}(1)$ parallel time, so the reduction algorithm is in NC.

Use this space for leftover answers and rough work

Use this space for leftover answers and rough work

Use this space for leftover answers and rough work

Use this space for leftover answers and rough work